Q1. Explain the difference between greedy and non-greedy syntax with visual terms in as few words as possible. What is the bare minimum effort required to transform a greedy pattern into a non-greedy one? What characters or characters can you introduce or change?

Ans1- Greedy: Matches as much as possible (e.g., .\* matches the longest string).

Non-greedy (or lazy): Matches as little as possible (e.g., .\*? matches the shortest string).

To transform a greedy pattern into a non-greedy one, add ? after quantifiers.

Q2. When exactly does greedy versus non-greedy make a difference?  What if you're looking for a non-greedy match but the only one available is greedy?

Ans2- Difference: Greedy captures more, non-greedy captures less.

Importance: Matters when extracting text between delimiters or in complex patterns. If only greedy is available, you may need to use additional logic to achieve non-greedy behavior.

Q3. In a simple match of a string, which looks only for one match and does not do any replacement, is the use of a nontagged group likely to make any practical difference?

Ans3- Non-tagged group in a simple match:

In a simple match without replacement, a non-tagged group ((?:...)) usually doesn't make a practical difference. Non-tagged groups are useful for grouping without capturing, saving memory when you don't need to reference the matched content.

Q4. Describe a scenario in which using a nontagged category would have a significant impact on the program's outcomes.

Ans4- Scenario for non-tagged category impact:

When you want to apply a quantifier to a group but don't want to capture its content, non-tagged groups (?:...) are essential. For example, in (\d+)(?:st|nd|rd|th), you capture numbers but not the ordinals, saving memory and simplifying access to the matched numbers.

Q5. Unlike a normal regex pattern, a look-ahead condition does not consume the characters it examines. Describe a situation in which this could make a difference in the results of your programme.

Ans5- Look-ahead condition not consuming characters:

This can make a difference when you want to match a pattern only if it's followed by or not followed by something else. For example, foo(?=bar) matches "foo" only if it's followed by "bar," allowing conditional matching.

Q6. In standard expressions, what is the difference between positive look-ahead and negative look-ahead?

Ans6- Positive look-ahead vs. negative look-ahead:

Positive look-ahead (?=...) asserts that a pattern must be present ahead.

Negative look-ahead (?!...) asserts that a pattern must not be present ahead.

Q7. What is the benefit of referring to groups by name rather than by number in a standard expression?

Ans7- Improved readability: Named groups make your regular expressions more self-explanatory.

Easier access: You can reference captured content by name, enhancing code clarity.

Maintainability: Renaming groups is easier without changing reference numbers.

Q8. Can you identify repeated items within a target string using named groups, as in "The cow jumped over the moon"?

Ans8- Identifying repeated items with named groups:

Yes, you can use named groups to find repeated items. For example, in the regex (?P<word>\w+)\s+(?P=word), it captures and identifies repeated words in a string.

Q9. When parsing a string, what is at least one thing that the Scanner interface does for you that the re.findall feature does not?

Ans9- The Scanner interface is not a feature in Python. In Python, re.findall is commonly used to find all non-overlapping matches in a string. It returns a list of matches. Unlike a Scanner, it doesn't provide methods for iterating over different parts of the input.

Q10. Does a scanner object have to be named scanner?

Ans10- No, a scanner object does not have to be named "scanner." You can choose any valid variable name that follows the naming rules in the programming language you are using.